10장. 객체 리터럴

10장. 객체 리터럴

10.1 객체란?

1. 자바스크립트는 **객체**(object) 기반의 프로그래밍 언어이며,

   자바스크립트를 구성하는 거의 "모든 것"이 객체.

   - 원시 값을 제외한 나머지 값(함수, 배열, 정규 표현식 등)은 모두 객체다.

2. **원시 타입의 값, 즉 원시 값은 변경 불가능한 값(immutable value)이지만,**

   객체 타입의 값, 즉 객체는 변경 가능한 값(mutable value)이다.

3. 객체는 0개 이상의 프로퍼티로 구성된 집합, 프로퍼티는 키(key) + 값(value)로 구성

```
var person = {
    name : 'Lee',   // 프로퍼티
    age : 20        // 프로퍼티
}
```

4. 자바스크립트에서 사양할 수 있는 모든 값은 프로퍼티 값이 될 수 있다.

   - 함수는 일급 객체 이므로 값으로 취급할 수 있다.

   ➡️ 따라서 함수도 프로퍼티 값으로 사용할 수 있다.

   - 프로퍼티 값이 함수일 경우, 일반 함수와 구분하기 위해 **메서드(method)**라 부른다.

5. **객체 = 프로퍼티 + 메서드**로 구성된 집합체

   - **프로퍼티** : 객체의 상태를 나타내는 값(data)

- **메서드** : 프로퍼티(상태 데이터)를 참조하고 조작할 수 있는 동작(behavior)

10.2 객체 리터럴에 의한 객체 생성

1. new 연산자와 함께 생성자(constructor)를 호출하여 인스턴스를 생성하는 방식으로 객체를 생성

2. 자바스크립트는 클래스 기반 객체지향 언어와 달리 다양한 객체 생성 방법이 있음.
   - 객체 리터럴
   - Object 생성자 함수
   - 생성자 함수
   - Object.create 메서드
   - 클래스(ES6)

3. 객체 리터럴은, 중괄호({...}) 내 0개 이상의 프로퍼티를 정의.

   변수가 할당되는 시점에 자바스크립트 엔진은 객체 리터럴을 해석해 객체를 생성.

   [예제 10 - 01]

```
var person = {
    name : 'Lee',
    sayHello : function() {
        console.log(`Hello! My name is ${this.name}.`);
    }
};

console.log(typeof person); // object
console.log(person);        // {name: 'Lee', sayHello: f}
```

4. 만약 중괄호 내에 프로퍼티를 정의하지 않으면, 빈 객체 생성

   [예제 10 - 02]

```
var empty = {};              // 빈 객체
console.log(typeof empty);   // object
```

10.3 프로퍼티

1. **객체 : 프로퍼티의 집합, 프로퍼티 : 키 + 값으로 구성**

2. **프로퍼티 키** : 빈 문자열을 포함하는 모든 문자열 or 심벌 값

   프로퍼티 값 : 자바스크립트에서 사용할 수 있는 모든 값

   - 프로퍼티 키 : **식별자 네이밍 규칙을 따르지 않는 이름에는 반드시 따옴표를 사용해야 한다.**

3. 프로퍼티 키에 문자열이나 심벌 값 외의 값을 사용하면 암묵적 타입 변환을 통해 문자열이 된다.

   [예제 10 - 08]

```
var foo = {
    0 : 1,
    1 : 2,
    2 : 3
};

console.log(foo);  // { 0:1, 1:2, 2:3 }
```

10.4 메서드

1. 프로퍼티 값이 함수일 경우 일반 함수와 구분하기 위해 메서드(method)라 부른다.

   즉, 메서드는 객체에 묶여 있는 함수를 의미한다.

[예제 10 - 11]

```
var Circle = {
    radius = 5, // <- 프로퍼티

    // 원의 지름
    getDiameter: function() {  // <- 메서드
        return 2 * this.radius;  // this는 circle 가리킴
    }
};

console.log(circle.getDiameter());  // 10
```

10.5 프로퍼티 접근

1. 프로퍼티에 접근하는 방법은 다음과 같이 두 가지다.
    - 마침표 프로퍼티 접근 연산자(.)를 사용하는 **마침표 표기법(dot notation)**
    - 대괄호 프로퍼티 접근 연산자([...])를 사용하는 **대괄호 표기법(bracket notation)**

2. [예제 10 - 12]

```
var person = {
    name : 'Lee'
};


// 마침표에 의한 프로퍼티 접근
console.log(person.name);     // Lee

// 대괄호에 의한 프로퍼티 접근
console.log(person['name']);  // Lee
```

대괄호 표기법 사용하는 경우, **대괄호 프로퍼티 접근 연산자 내부에 지정하는 프로퍼티 키는 반드시 따옴표로 감싼 문자열**이어야 한다.

10.6 프로퍼티 값 갱신

1. 이미 존재하는 프로퍼티에 값을 할당하면 프로퍼티 값이 갱신된다.

10.7 프로퍼티 동적 생성

1. 존재하지 않는 프로퍼티에 값을 할당하면,

   프로퍼티가 동적으로 생성되어 추가되고 프로퍼티 값이 할당됨.

10.8 프로퍼티 삭제

1. [예제 10 - 18]

```
var person = {
    name : 'Lee'
};

person.age = 20; // 프로퍼티 동적 생성

delete person.age;
```

10.9 ES6에서 추가된 객체 리터럴의 확장 기능

10.9.1 프로퍼티 축약 표현

1. ES6에서는 프로퍼티 값으로 변수를 사용하는 경우,

   변수 이름과 프로퍼티 키가 동일한 이름일 때 **프로퍼티 키를 생략(property shorthand)**할 수 있다.

   [예제 10 - 20]

```
let x = 1, y = 2;

// 프로퍼티 축약 표현
const obj = { x, y };

console.log(obj);  // {x:1, y:2}
```

10.9.2 계산된 프로퍼티 이름

10.9.3 메서드 축약 표현

1. ES5에서 메서드를 정의하려면 프로퍼티 값으로 함수를 할당한다.

2. ES6에서는 메서드를 정의할 때 function 키워드를 생략한 축약 표현을 사용할 수 있다.

   [예제 10 - 24]

```
// ES6
const obj{
    name : 'Lee',
    // 메서드 축약 표현
    sayHi(){
        console.log('Hi' + this.name);
    }
};

obj.sayHi(0;  // Hi! Lee
```